Part 1

Primitives: The ingredients of cryptography

Welcome to the real-world of cryptography! The book you're holding in your hands (if you chose to acquire a printed version) is split into two equal parts of eight chapters. By going through all of it, you will learn (almost) all there is to know about cryptography in the real world—the one you're standing in.

Note that the first part of the book was written to be read in order, although each chapter should tell you what the prerequisites are, so do not view this as a mandatory constraint. The first eight chapters take you through the basics—the building blocks of cryptography. Each chapter introduces a new ingredient and teaches you what it does, how it works, and how it can be used with other elements. This first part is all about giving you good abstractions and insights before we start making use of it all in the second part of the book.

Good luck!

1 Introduction

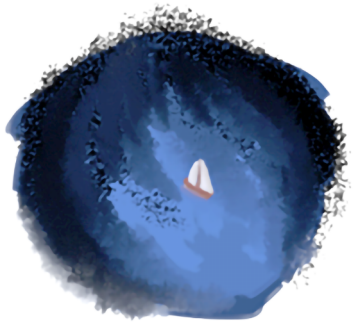
This chapter covers

- What cryptography is about
- Theoretical versus real-world cryptography
- What you'll learn throughout this adventure

Greetings, traveler; sit tight. You're about to enter a world of wonder and mystery—the world of cryptography. *Cryptography* is the ancient discipline of securing situations that are troubled with malicious characters. This book includes the spells that we need to defend ourselves against the malice. Many have attempted to learn this craft, but few have survived the challenges that stand in the way of mastery. Exciting adventures await, indeed!

In this book, we'll uncover how cryptographic algorithms can secure our letters, identify our allies, and protect treasures from our enemies. Sailing through the cryptographic sea will not be the smoothest journey as cryptography is the foundation of all security and privacy in our world—the slightest mistake could be deadly.

NOTE If you find yourself lost, remember to keep moving forward. It will all eventually make sense.



1.1 Cryptography is about securing protocols

Our journey starts with an introduction to cryptography, the science aiming to defend protocols against saboteurs. But first, what's a *protocol*? Simply put, it's a list of steps that one (or more people) must follow in order to achieve something. For example, imagine the following premise: you want to leave your magic sword unattended for a few hours so you can take a nap. One protocol to do this could be the following:

1. Deposit weapon on the ground
2. Take nap under a tree
3. Recover weapon from the ground

Of course, it's not a great protocol as anybody can steal your sword while you're napping . . . And so, cryptography is about taking into account the adversaries who are looking to take advantage of you.

In ancient times, when rulers and generals were busy betraying each other and planning coups, one of their biggest problems was finding a way to *share confidential information with those they trusted*. From here, the idea of cryptography was born. It took centuries and hard work before cryptography became the serious discipline it is today. Now, it's used all around us to provide the most basic services in the face of our chaotic and adverse world.

The story of this book is about the practice of cryptography. It takes you on an expedition throughout the computing world to cover cryptographic protocols in use today; it also shows you what parts they are made of and how everything fits together. While a typical cryptography book usually starts with the discovery of cryptography and takes you through its history, I think that it makes little sense for me to kick off things that way. I want to tell you about the practical. I want to tell you about what I've witnessed myself, reviewing cryptographic applications for large companies as a consultant, or the cryptography I've made use of myself as an engineer in the field.

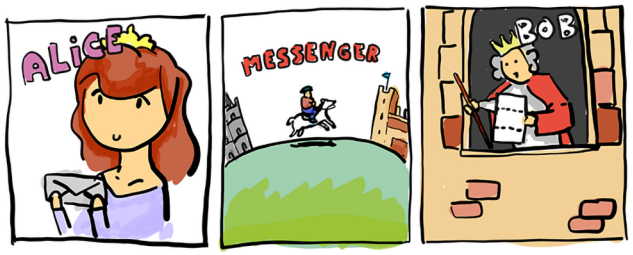
There will be (almost) no scary math formulas. The purpose of this book is to demystify cryptography, survey what is considered useful nowadays, and provide intuition about how things around you are built. This book is intended for curious people, interested engineers, adventurous developers, and inquisitive researchers. Chapter 1,

this chapter, initiates a tour of the world of cryptography. We will discover the different types of cryptography, which ones matter to us, and how the world agreed on using these.

1.2 Symmetric cryptography: What is symmetric encryption?

One of the fundamental concepts of cryptography is *symmetric encryption*. It is used in a majority of cryptographic algorithms in this book, and it is, thus, extremely important. I introduce this new concept here via our first protocol.

Let's imagine that Queen Alice needs to send a letter to Lord Bob, who lives a few castles away. She asks her loyal messenger to ride his trusty steed and battle his way through the dangerous lands ahead in order to deliver the precious message to Lord Bob. Yet, she is suspicious; even though her loyal messenger has served her for many years, she wishes the message in transit to remain secret from all passive observers, including the messenger! You see, the letter most likely contains some controversial gossip about the kingdoms on the way.



What Queen Alice needs is a protocol that mimics handing the message to Lord Bob herself with no middlemen. This is quite an impossible problem to solve in practice unless we introduce cryptography (or teleportation) into the equation. This is what we ended up doing ages ago by inventing a new type of cryptographic algorithm—called a *symmetric encryption algorithm* (also known as a *cipher*).

> **NOTE** By the way, a type of cryptographic algorithm is often referred to as a *primitive*. You can think of a primitive as the smallest, useful construction you can have in cryptography, and it is often used with other primitives in order to build a protocol. It is mostly a term and has no particularly important meaning, although it appears often enough in the literature that it is good to know about it.

Let's see how we can use an encryption primitive to hide Queen Alice's message from the messenger. Imagine for now that the primitive is a black box (we can't see what's inside or what it's doing internally) that provides two functions:

- ENCRYPT
- DECRYPT

The first function, ENCRYPT, works by taking a *secret key* (usually a large number) and a *message*. It then outputs a series of random-looking numbers, some noisy data if you will. We will call that output the encrypted message. I illustrate this in figure 1.1.

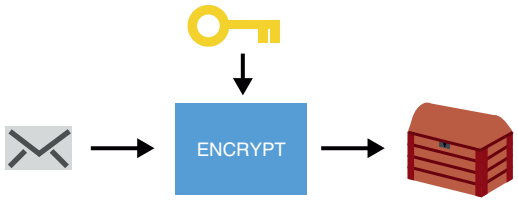


Figure 1.1 The ENCRYPT function takes a message and a secret key and outputs the encrypted message—a long series of numbers that look like random noise.

The second function, DECRYPT, is the inverse of the first one. It takes the same secret key and the random output of the first function (the encrypted message) and then it finds the original message. I illustrate this in figure 1.2.

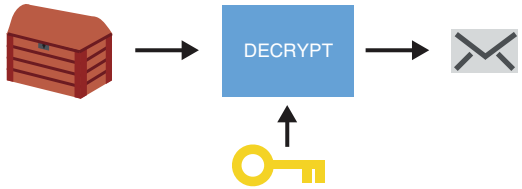


Figure 1.2 The DECRYPT function takes an encrypted message and a secret key and returns the original message.

To make use of this new primitive, Queen Alice and Lord Bob have to first meet in real life and decide on what secret key to use. Later, Queen Alice can use the provided ENCRYPT function to protect a message with the help of the secret key. She then passes the encrypted message to her messenger, who eventually delivers it to Lord Bob. Lord Bob then uses the DECRYPT function on the encrypted message with the same secret key to recover the original message. Figure 1.3 shows this process.

During this exchange, all the messenger had was something that looked random and that provided no meaningful insight into the content of the hidden message. Effectively, we augmented our insecure protocol into a secure one, thanks to the help of cryptography. The new protocol makes it possible for Queen Alice to deliver a confidential letter to Lord Bob without anyone (except Lord Bob) learning the content of it.

The process of using a secret key to render things to noise, making them indistinguishable from random, is a common way of securing a protocol in cryptography. You will see more of this as you learn more cryptographic algorithms in the next chapters.

By the way, symmetric encryption is part of a larger category of cryptography algorithms called *symmetric cryptography* or *secret key cryptography*. This is due to the same key being used by the different functions exposed by the cryptographic primitive. As you will see later, sometimes there's more than one key.

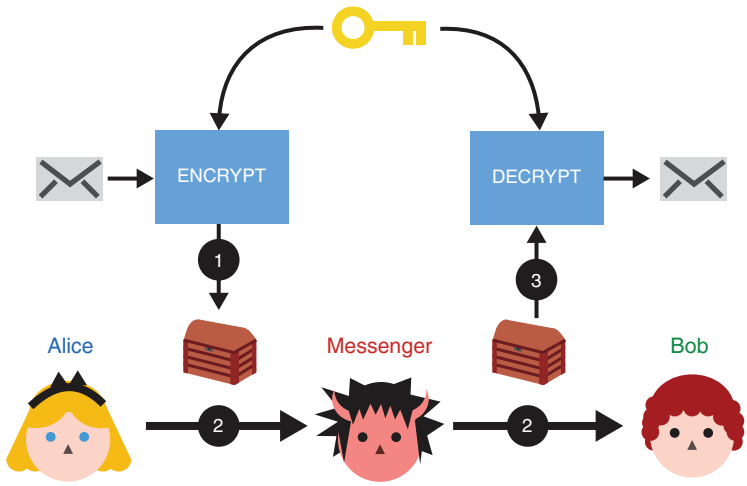


Figure 1.3 (1) Alice uses the ENCRYPT function with a secret key to transform her message into noise. (2) She then passes the encrypted message to her messenger, who will not learn anything about the underlying message. (3) Once Bob receives the encrypted message, he can recover the original content by using the DECRYPT function with the same secret key Alice used.

1.3 Kerckhoff's principle: Only the key is kept secret

To design a cryptographic algorithm (like our encryption primitive) is an easy task, but to design a *secure* cryptographic algorithm is not for the faint of heart. While we shy away from creating such algorithms in this book, we *do* learn how to recognize the good ones. This can be difficult as there is more choice than one can ask for the task. Hints can be found in the repeated failures of the history of cryptography, as well as the lessons that the community has learned from them. As we take a look at the past, we will grasp at what turns a cryptographic algorithm into a trusted-to-be-secure one.

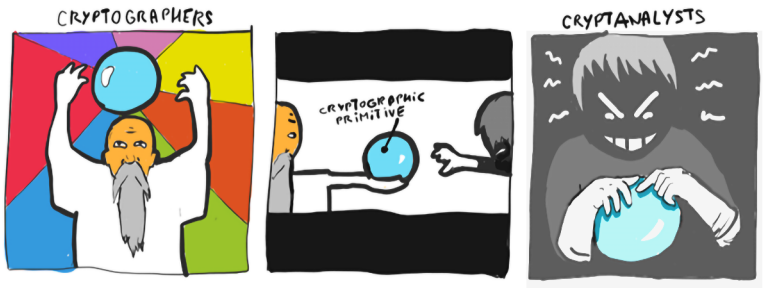
Hundreds of years have passed and many queens and lords have been buried. Since then, paper has been abandoned as our primary means of communication in favor of better and more practical technologies. Today, we have access to powerful computers as well as the internet. More practical, sure, but this also means that our previous malicious messenger has become much more powerful. He is now everywhere: the Wi-Fi in the Starbucks cafe you're sitting in, the different servers making up the internet and forwarding your messages, and even in the machines running our algorithms. Our enemies are now able to observe many more messages as each request you make to a website might pass through the wrong wire and become altered or copied in a matter of nanoseconds without anyone noticing.

Before us, we can see that recent history contains many instances of encryption algorithms falling apart, being broken by secret state organizations or by independent researchers, and failing to protect their messages or accomplish their claims. Many

lessons were learned, and we slowly came to understand how to produce good cryptography.

> **NOTE** A cryptographic algorithm can be considered *broken* in many ways. For an encryption algorithm, you can imagine several ways to attack the algorithm: the secret key can be leaked to the attacker, messages can be decrypted without the help of the key, some information about the message can be revealed just by looking at the encrypted message, and so on. Anything that would somehow weaken the assumptions we made about the algorithm could be considered a break.

A strong notion came out of the long process of trial and error that cryptography went through: to obtain confidence in the security claims made by a cryptographic primitive, the primitive has to be analyzed in the open by experts. Short of that, you are relying on *security through obscurity*, which hasn't worked well historically. This is why *cryptographers* (the people who build) usually use the help of *cryptanalysts* (the people who break) in order to analyze the security of a construction. (Although cryptographers are often cryptanalysts themselves and vice-versa.)



Let's take the Advanced Encryption Standard (AES) encryption algorithm as an example. AES was the product of an international competition organized by the National Institute of Standards and Technology (NIST).

> **NOTE** NIST is a United States agency whose role is to define standards and develop guidelines for use in government-related functions as well as other public or private organizations. Like AES, it has standardized many widely used cryptographic primitives.

The AES competition lasted several years, during which many volunteering cryptanalysts from around the world gathered to take a chance at breaking the various candidate constructions. After several years, once enough confidence was built by the process, a single competing encryption algorithm was nominated to become the Advanced

Encryption Standard itself. Nowadays, most people trust that AES to be a solid encryption algorithm, and it is widely used to encrypt almost anything. For example, you use it every day when you browse the web.

The idea to build cryptographic standards in the open is related to a concept often referred to as *Kerckhoffs' principle*, which can be understood as something like this: it would be foolish to rely on our enemies not to discover what algorithms we use because they most likely will. Instead, let's be open about them.

If the enemies of Queen Alice and Lord Bob knew exactly how they were encrypting messages, how is their encryption algorithm secure? The answer is the *secret key*! The secrecy of the key makes the protocol secure, not the secrecy of the algorithm itself. This is a common theme in this book: all the cryptographic algorithms that we will learn about and that are used in the real world are most often free to be studied and used. Only the secret keys used as input to these algorithms are kept secret. *Ars ipsi secreta magistro* (an art secret even for the master), said Jean Robert du Carlet in 1644. In the next section, I will talk about a totally different kind of cryptographic primitive. For now, let's use figure 1.4 to organize what we've learned so far.

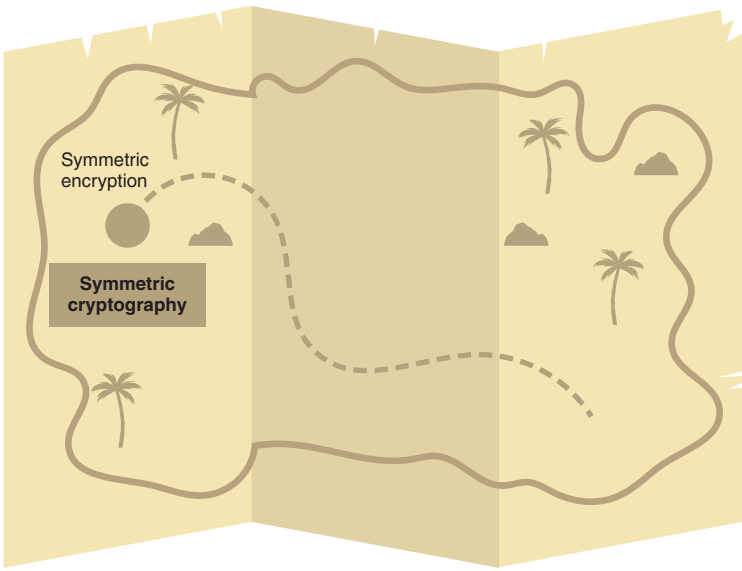


Figure 1.4 The cryptographic algorithms you have learned so far. AES is an instantiation of a symmetric encryption algorithm, which is a cryptographic primitive that is part of the broader class of symmetric cryptographic algorithms.

1.4 Asymmetric cryptography: Two keys are better than one

In our discussion about symmetric encryption, we said that Queen Alice and Lord Bob first met to decide on a symmetric key. This is a plausible scenario, and a lot of protocols actually do work like this. Nonetheless, this quickly becomes less practical in protocols with many participants: do we need our web browser to meet with Google, Facebook, Amazon, and the other billions of websites before securely connecting to those?

This problem, often referred to as *key distribution*, has been a hard one to solve for quite a long time, at least until the discovery in the late 1970s of another large and useful category of cryptographic algorithms called *asymmetric cryptography* or *public key cryptography*. Asymmetric cryptography generally makes use of different keys for different functions (as opposed to a single key used in symmetric cryptography) or provides different points of view to different participants. To illustrate what this means and how public key cryptography helps to set up trust between people, I'll introduce a number of asymmetric primitives in this section. Note that this is only a glance of what you'll learn in this book as I'll talk about each of these cryptographic primitives in more detail in subsequent chapters.

1.4.1 Key exchanges or how to get a shared secret

The first asymmetric cryptography primitive we'll look at is the *key exchange*. The first public key algorithm discovered and published was a key exchange algorithm named after its authors, Diffie-Hellman (DH). The DH key exchange algorithm's main purpose is to establish a common secret between two parties. This common secret can then be used for different purposes (for example, as a key to a symmetric encryption primitive).

In chapter 5, I will explain how Diffie-Hellman works, but for this introduction, let's use a simple analogy in order to understand what a key exchange provides. Like many algorithms in cryptography, a key exchange must start with the participants using a common set of parameters. In our analogy, we will simply have Queen Alice and Lord Bob agree to use a square (■). The next step is for them to choose their own random shape. Both of them go to their respective secret place, and out of sight, Queen Alice chooses a triangle (▲) and Lord Bob chooses a star (★). The objects they chose need to remain secret at all costs! These objects represent their *private keys* (see figure 1.5).

Once they chose their private keys, they both individually combine their secret shape with the common shape they initially agreed on using (the square). The combi-

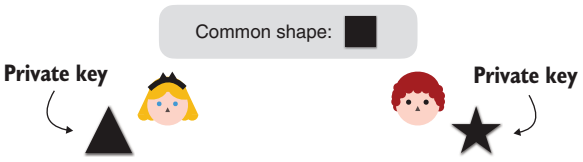


Figure 1.5 The first step of a DH (Diffie-Hellman) key exchange is to have both participants generate a private key. In our analogy, Queen Alice chooses a triangle as her private key, whereas Lord Bob chooses a star as his private key.

nations result in unique shapes representing their *public keys*. Queen Alice and Lord Bob can now exchange their public keys (hence the name *key exchange*) because public keys are considered public information. I illustrate this in figure 1.6.

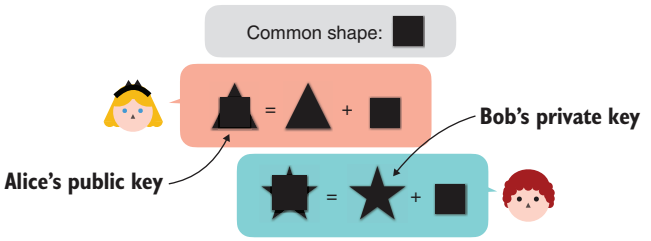


Figure 1.6 The second step of a DH key exchange where both participants exchange their public keys. Participants derive their public keys by combining their private keys with a common shape.

We are now starting to see why this algorithm is called a public key algorithm. It is because it requires a *key pair* comprised of a private key and a public key. The final step of the DH key exchange algorithm is quite simple: Queen Alice takes Lord Bob's public key and combines it with her private key. Lord Bob does the same with Queen Alice's public key and combines it with his own private key. The result should now be the same on each side; in our example, a shape combining a star, a square, and a triangle (see figure 1.7).

Figure 1.7 The final step of a DH key exchange where both participants produce the same shared secret. To do this, Queen Alice combines her private key with Lord Bob's public key, and Lord Bob combines his private key with Queen Alice's public key. The shared secret cannot be obtained from solely observing the public keys.

It is now up to the participants of the protocol to make use of this shared secret. You will see several examples of this in this book, but the most obvious scenario is to make use of it in an algorithm that requires a shared secret. For example, Queen Alice and Lord Bob could now use the shared secret as a key to encrypt further messages with a symmetric encryption primitive. To recap

1. Alice and Bob exchange their public keys, which masks their respective private keys.
2. With the other participant's public key and their respective private key, they can compute a shared secret.
3. An adversary who observes the exchange of public keys doesn't have enough information to compute the shared secret.

NOTE In our example, the last point is easily bypassable. Indeed, without the knowledge of any private keys, we can combine the public keys together to produce the shared secret. Fortunately, this is only a limitation of our analogy, but it works well enough for us to understand what a key exchange does.

In practice, a DH key exchange is quite insecure. Can you take a few seconds to figure out why?

Because Queen Alice accepts any public key she receives as being Lord Bob's public key, I could intercept the exchange and replace it with mine, which would allow me to impersonate Lord Bob to Queen Alice (and the same can be done to Lord Bob). We say that a *man-in-the-middle* (MITM) attacker can successfully attack the protocol. How do we fix this? We will see in later chapters that we either need to augment this protocol with another cryptographic primitive, or we need to be aware in advance of what Lord Bob's public key is. But then, aren't we back to square one?

Previously, Queen Alice and Lord Bob needed to know a shared secret; now Queen Alice and Lord Bob need to know their respective public keys. How do they get to know that? Is that a chicken-and-egg problem all over again? Well, kind of. As we will see, in practice, public key cryptography does not solve the problem of trust, but it simplifies its establishment (especially when the number of participants is large).

Let's stop here and move on to the next section as you will learn more about key exchanges in chapter 5. We still have a few more asymmetric cryptographic primitives to uncover (see figure 1.8) to finish our tour of real-world cryptography.

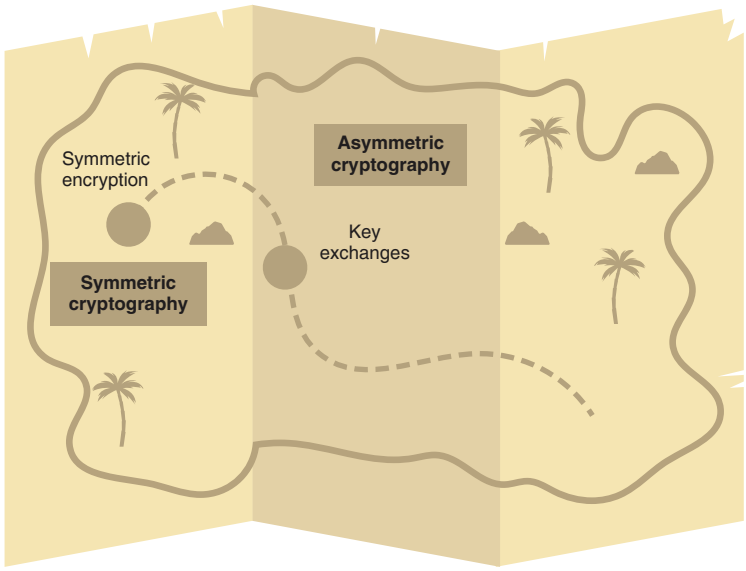


Figure 1.8 The cryptographic algorithms we have learned so far. Two large classes of cryptographic algorithms are symmetric cryptography (with symmetric encryption) and asymmetric cryptography (with key exchanges).

1.4.2 Asymmetric encryption, not like the symmetric one

The invention of the DH key exchange algorithm was quickly followed by the invention of the *RSA algorithm* named after Ron Rivest, Adi Shamir, and Leonard Adleman. RSA contains two different primitives: a public key encryption algorithm (or asymmetric encryption) and a (digital) signature scheme. Both primitives are part of the larger class of cryptographic algorithms called *asymmetric cryptography*. In this section, we will explain what these primitives do and how they can be useful.

The first one, asymmetric encryption, has a similar purpose to the symmetric encryption algorithm we talked about previously: it allows one to encrypt messages in order to obtain confidentiality. Yet, unlike symmetric encryption, which had the two participants encrypt and decrypt messages with the same symmetric key, asymmetric encryption is quite different:

- It works with two different keys: a public key and a private key.
- It provides an asymmetric point of view: anyone can encrypt with the public key, but only the owner of the private key can decrypt messages.

Let's now use a simple analogy to explain how one can use asymmetric encryption. We start with our friend Queen Alice again, who holds a private key (and its associated public key). Let's picture her public key as an open chest that she releases to the public for anyone to use (see figure 1.9).

Figure 1.9 To use asymmetric encryption, Queen Alice needs to first publish her public key (represented as an open box here). Now, anyone can use the public key to encrypt messages to her. And she should be able to decrypt them using the associated private key.

Now, you and I and everyone who wants can encrypt a message to her using her public key. In our analogy, imagine that you would insert your message into the open chest and then close it. Once the chest is closed, nobody but Queen Alice should be able to open it. The box effectively protects the secrecy of the message from observers. The closed box (or encrypted content) can then be sent to Queen Alice, and she can use her private key (only known to her, remember) to decrypt it (see figure 1.10).

Let's summarize in figure 1.11 the cryptographic primitives we have learned so far. We are only missing one more to finish our tour of real-world cryptography!

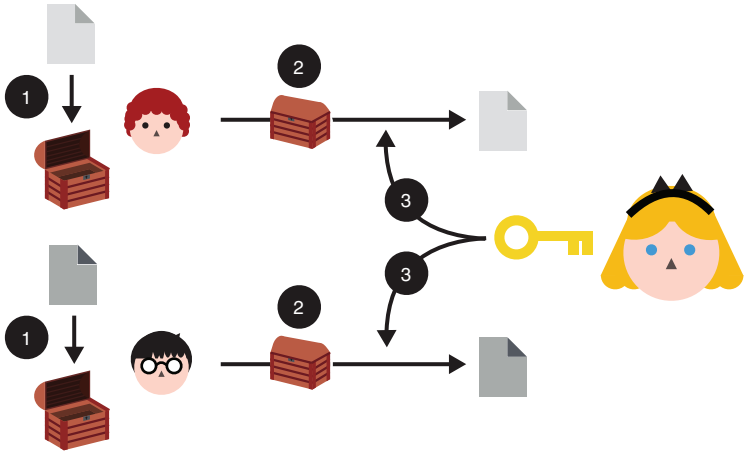


Figure 1.10 Asymmetric encryption: (1) anyone can use Queen Alice's public key to encrypt messages to her. (2) After receiving them, (3) she can decrypt the content using her associated private key. Nobody is able to observe the messages directed to Queen Alice while they are being sent to her.

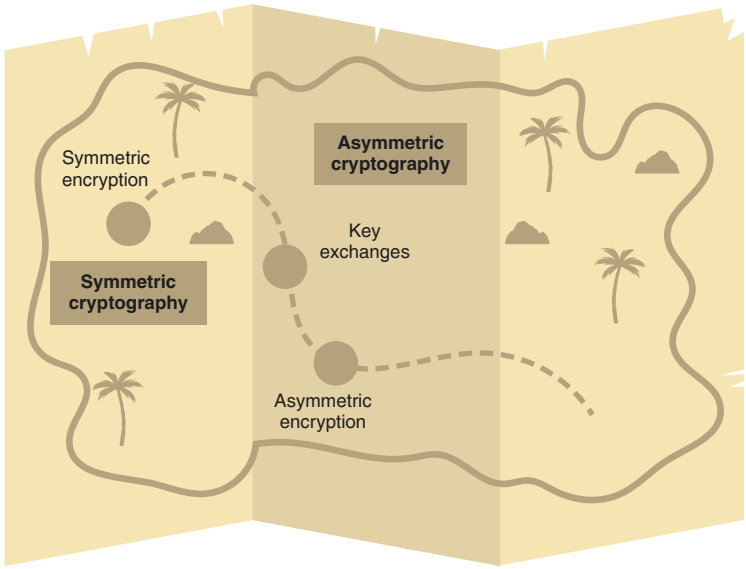


Figure 1.11 The cryptographic algorithms we have learned so far: two large classes of cryptographic algorithms are symmetric cryptography (with symmetric encryption) and asymmetric cryptography (with key exchanges and asymmetric encryption).

1.4.3 Digital signatures, just like your pen-and-paper signatures

We saw that RSA provides an asymmetric encryption algorithm, but as we mentioned earlier, it also provides a *digital signature* algorithm. The invention of this digital signature cryptographic primitive has been of immense help to set up trust between the Alices and Bobs of our world. It is similar to real signatures; you know, the one that you are required to sign on a contract when you're trying to rent an apartment, for example.

"What if they forge my signature?" you may ask, and indeed, real signatures don't provide much security in the real world. On the other hand, cryptographic signatures can be used in the same kind of way but provide a cryptographic certificate with your name on it. Your cryptographic signature is *unforgeable* and can easily be verified by others. Pretty useful compared to the archaic signatures you used to write on checks!

In figure 1.12, we can imagine a protocol where Queen Alice wants to show Lord David that she trusts Lord Bob. This is a typical example of how to establish trust in a multiparticipant setting and how asymmetric cryptography can help. By signing a piece of paper containing "I, Queen Alice, trust Lord Bob," Queen Alice can take a stance and notify Lord David that Lord Bob is to be trusted. If Lord David already trusts Queen Alice and her signature algorithm, then he can choose to trust Lord Bob in return.

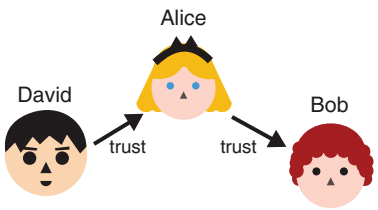


Figure 1.12 Lord David already trusts Queen Alice. Because Queen Alice trusts Lord Bob, can Lord David safely trust Lord Bob as well?

In more detail, Queen Alice can use the RSA signature scheme and her private key to sign the message, "I, Queen Alice, trust Lord Bob." This generates a signature that should look like random noise (see figure 1.13).

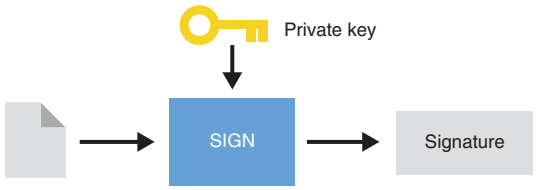


Figure 1.13 To sign a message, Queen Alice uses her private key and generates a signature.

Anyone can then *verify the signature* by combining:

- Alice's public key
- The message that was signed
- The signature

The result is either *true* (the signature is valid) or *false* (the signature is invalid) as figure 1.14 shows.

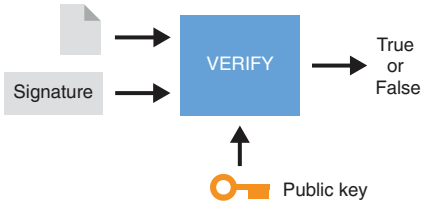


Figure 1.14 To verify a signature from Queen Alice, one also needs the message signed and Queen Alice's public key. The result is either validating the signature or invalidating it.

We have now learned about three different asymmetric primitives:

- Key exchange with Diffie-Hellman
- Asymmetric encryption
- Digital signatures with RSA

These three cryptographic algorithms are the most known and commonly used primitives in asymmetric cryptography. It might not be totally obvious how they can help to solve real-world problems, but rest assured, they are used every day by many applications to secure things around them. It is time to complete our picture with all the cryptographic algorithms we've learned about so far (see figure 1.15).

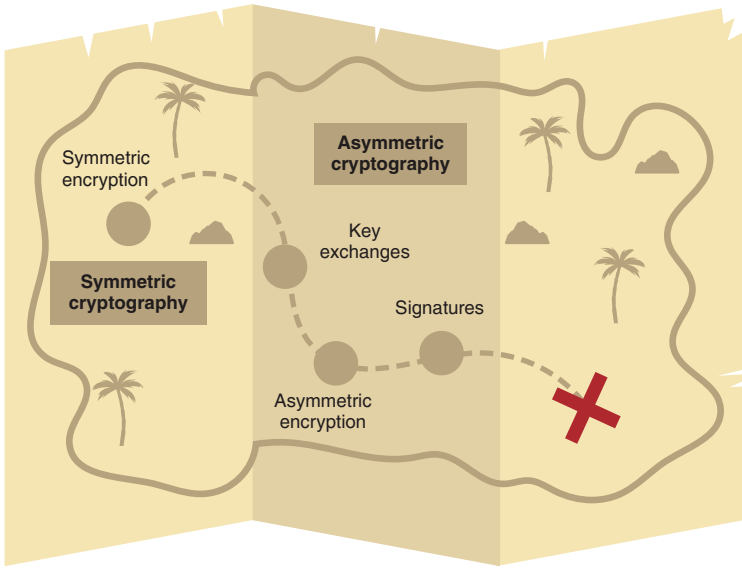


Figure 1.15 The symmetric and asymmetric algorithms we have learned so far

1.5 Classifying and abstracting cryptography

In the previous section, we surveyed two large classes of algorithms:

- *Symmetric cryptography (or secret key cryptography)*—A single secret is used. If several participants are aware of the secret, it is called a *shared* secret.
- *Asymmetric cryptography (or public key cryptography)*—Participants have an asymmetrical view of the secrets. For example, some will have knowledge of a public key, while some will have knowledge of both a public and private key.

Symmetric and asymmetric cryptography are not the only two categories of primitives in cryptography, and it's quite hard to classify the different subfields. But yet, as you will realize, a large part of our book is about (and makes use of) symmetric and asymmetric primitives. This is because a large part of what is useful in cryptography nowadays is contained in these subfields. Another way of dividing cryptography can be

- *Math-based constructions*—These rely on mathematical problems like factoring numbers. (The RSA algorithm for digital signatures and asymmetric encryption is an example of such a construction.)
- *Heuristic-based constructions*—These rely on observations and statistical analysis by cryptanalysts. (AES for symmetric encryption is an example of such a construction.)

There is also a speed component to this categorization as mathematic-based constructions are often much slower than heuristic-based constructions. To give you an idea, symmetric constructions are most often based on heuristics (what seems to be working), while most asymmetric constructions are based on mathematical problems (what is thought to be hard).

It is hard for us to rigorously categorize all of what cryptography has to offer. Indeed, every book or course on the subject gives different definitions and classifications. In the end, these distinctions are not too useful for us as we will see most of the cryptographic primitives as unique tools that make unique *security claims*. We can, in turn, use many of these tools as building blocks to create protocols. It is thus essential to understand how each of these tools work and what kind of security claims they provide in order to understand how they secure the protocols around us. For this reason, the first part of this book will go through the most useful cryptographic primitives and their security properties.

A lot of the concepts in the book can be quite complicated the first time around. But like everything, the more we read about them and the more we see them in context, the more natural they become, the more we can abstract them. The role of this book is to help you to create abstractions, to allow you to create a mental model of what these constructions do, and to understand how they can be combined together to produce secure protocols. I will often talk about the interface of constructions and give real-world examples of usage and composition.

The definition of cryptography used to be simple: Queen Alice and Lord Bob want to exchange secret messages. It isn't anymore. What cryptography is nowadays is quite

complex to describe and has grown organically around discoveries, breakthroughs, and practical needs. At the end of the day, cryptography is what helps to augment a protocol in order to make it work in adversarial settings.

To understand exactly how cryptography can help, the set of goals that these protocols aim to achieve is what matters to us. That's the useful part. Most of the cryptographic primitives and protocols we'll learn about in this book provide one or two of the following properties:

- *Confidentiality*—It's about masking and protecting some information from the wrong eyes. For example, encryption masks the messages in transit.
- *Authentication*—It's about identifying who we are talking to. For example, this can be helpful in making sure that messages we receive indeed come from Queen Alice.

Of course, this is still a heavy simplification of what cryptography can provide. In most cases, the details are in the security claims of the primitives. Depending on how we use a cryptographic primitive in a protocol, it will achieve different security properties.

Throughout this book, we will learn new cryptographic primitives and how they can be combined to expose security properties like confidentiality and authentication. For now, appreciate the fact that cryptography is about providing insurances to a protocol in adversarial settings. While the "adversaries" are not clearly defined, we can imagine that they are the ones who attempt to break our protocol: a participant, an observer, a man in the middle. They reflect what a real-life adversary could be. Because eventually, cryptography is a practical field made to defend against bad actors in flesh and bones and bits.

1.6 Theoretical cryptography vs. real-world cryptography

In 1993, Bruce Schneier released *Applied Cryptography* (Wiley), a book targeting developers and engineers who want to build applications that involve cryptography. Circa 2012, Kenny Paterson and Nigel Smart started an annual conference called Real World Crypto that targets the same crowd. But what do applied cryptography and real-world cryptography refer to? Is there more than one type of cryptography?

To answer the questions, we have to start by defining *theoretical cryptography*, the cryptography that cryptographers and cryptanalysts work on. These crypto people are mostly from academia, working in universities, but sometimes from the industry or in specific departments of the government. They research everything and anything in cryptography. Results are shared internationally through publications and presentations in journals and conferences. Yet not everything they do is obviously useful or practical. Often, no "proof of concept" or code is released. It wouldn't make sense anyway, as no computer is powerful enough to run their research. Having said that, theoretical cryptography sometimes becomes so useful and practical that it makes its way to the other side.

The other side is the world of *applied cryptography* or *real-world cryptography*. It is the foundation of the security you find in all applications around you. Although it often seems like it's not there, almost transparent, it is there when you log into your bank account on the internet; it is with you when you message your friends; it helps protect you when you lose your phone. It is ubiquitous because, unfortunately, attackers are everywhere and actively try to observe and harm our systems. Practitioners are usually from the industry but will sometimes vet algorithms and design protocols with the help of the academic community. Results are often shared through conferences, blog posts, and open source software.

Real-world cryptography usually cares deeply about real-world considerations: what is the exact level of security provided by an algorithm? How long does it take to run the algorithm? What is the size of the inputs and outputs required by the primitive? Real-world cryptography is, as you might have guessed, the subject of this book. While theoretical cryptography is the subject of other books, we will still take a peek at what is brewing there in the last chapters of this book. Be prepared to be amazed as you might catch a glance of the real-world cryptography of tomorrow.

Now you might be wondering: how do developers and engineers choose what cryptography to use for their real-world applications?

1.7 From theoretical to practical: Choose your own adventure

> *Sitting on top are cryptanalysts who propose and solve hard mathematical problems [. . .] and at the bottom are software engineers who want to encrypt some data.*
>
> —Thai Duong ("So you want to roll your own crypto?," 2020)

In all the years I've spent studying and working with cryptography, I've never noticed a single pattern in which a cryptographic primitive ends up being used in real-world applications. Things are pretty chaotic. Before a theoretical primitive gets to be adopted, there's a long list of people who get to handle the primitive and shape it into something consumable and sometimes safer for the public at large. How can I even explain that to you?

Have you heard of *Choose Your Own Adventure*? It's an old book series where you got to pick how you want to step through the story. The principle was simple: you read the first section of the book; at the end of the section, the book lets you decide on the path forward by giving you different options. Each option was associated with a different section number that you could skip directly to if you so chose. So, I did the same here! Start by reading the next paragraph and follow the direction it gives you.

Where it all begins. Who are you? Are you Alice, a cryptographer? Are you David, working in the private industry and in need of a solution to your problems? Or are you Eve, working in a government branch and preoccupied by cryptography?

- You're Alice, go to step 1.
- You're David, go to step 2.
- You're Eve, go to step 3.

Step 1: Researchers gotta research. You're a researcher working in a university, or in the research team of a private company or a nonprofit, or in a government research organization like NIST or NSA. As such, your funding can come from different places and might incentivize you to research different things.

- You invent a new primitive, go to step 4.
- You invent a new construction, go to step 5.
- You start an open competition, go to step 6.

Step 2: The industry has a need. As part of your job, something comes up and you are in need of a new standard. For example, the Wi-Fi Alliance is a nonprofit funded by interested companies to produce the set of standards around the Wi-Fi protocol. Another example are banks that got together to produce the Payment Card Industry Data Security Standard (PCI-DSS), which enforces algorithms and protocols to use if you deal with credit card numbers.

- You decide to fund some much needed research, go to step 1.
- You decide to standardize a new primitive or protocol, go to step 5.
- You start an open competition, go to step 6.

Step 3: A government has a need. You're working for your country's government, and you need to push out some new crypto. For example, the NIST is tasked with publishing the *Federal Information Processing Standards* (FIPS), which mandates what cryptographic algorithms can be used by companies that deal with the US government. While many of these standards were success stories and people tend to have a lot of trust in standards being pushed by government agencies, there is (unfortunately) a lot to say about failures.

In 2013, following revelations from Edward Snowden, it was discovered that NSA had purposefully and successfully pushed for the inclusion of backdoor algorithms in standards (see "Dual EC: A Standardized Back Door" by Bernstein et al.), which included a hidden switch that allowed NSA, and only the NSA, to predict your secrets. These *backdoors* can be thought of as magic passwords that allow the government (and only it, supposedly) to subvert your encryption. Following this, the cryptographic community lost a lot of confidence in standards and suggestions coming from governmental bodies. Recently, in 2019, it was found that the Russian standard GOST had been a victim of the same treatment.

> *Cryptographers have long suspected that the agency planted vulnerabilities in a standard adopted in 2006 by the National Institute of Standards and Technology and later by the International Organization for Standardization, which has 163 countries as members. Classified N.S.A. memos appear to confirm that the fatal weakness, discovered by two Microsoft cryptographers in 2007, was engineered by the agency. The N.S.A. wrote the*

> *standard and aggressively pushed it on the international group, privately calling the effort "a challenge in finesse."*
>
> —*New York Times* ("N.S.A. Able to Foil Basic Safeguards of Privacy on Web," 2013)

- You fund some research, go to step 1.
- You organize an open competition, go to step 6.
- You push for the standardization of a primitive or protocol that you're using, go to step 7.

Step 4: A new concept is proposed. As a researcher, you manage to do the impossible; you invent a new concept. Sure, someone already thought about encryption, but there are still new primitives being proposed every year in cryptography. Some of them will prove to be impossible to realize, and some will end up being solvable. Maybe you have an actual construction as part of your proposal, or maybe you'll have to wait to see if someone can come up with something that works.

- Your primitive gets implemented, go to step 5.
- Your primitive ends up being impossible to implement, go back to the beginning.

Step 5: A new construction or protocol is proposed. A cryptographer or a team of cryptographers proposes a new algorithm that instantiates a concept. For example, AES is an instantiation of an encryption scheme. (AES was initially proposed by Vincent Rijmen and Joan Daemen, who named their construction as a contraction of their names, Rijndael.) What's next?

- Someone builds on your construction, go to step 5.
- You partake in an open competition and win! Go to step 6.
- There's a lot of hype for your work; you're getting a standard! Go to step 7.
- You decide to patent your construction, go to step 8.
- You or someone else decides that it'll be fun to implement your construction. Go to step 9.

Step 6: An algorithm wins a competition. The process cryptographers love the most is an open competition! For example, AES was a competition that invited researchers from all over the world to compete. After dozens of submissions and rounds of analysis and help from cryptanalysts (which can take years), the list was reduced to a few candidates (in the case of AES, a single one), which then moved to become standardized.

- You got lucky, after many years of competition your construction won! Go to step 7.
- Unfortunately, you lost. Go back to the start.

Step 7: An algorithm or protocol is standardized. A standard is usually published by a government or by a standardization body. The aim is to make sure that everyone is on the same page so as to maximize interoperability. For example, NIST regularly publishes cryptographic standards. A well-known standardization body in cryptography is the

Internet Engineering Task Force (IETF), which is behind many standards on the internet (like TCP, UDP, TLS, and so on) and that you will hear about a lot in this book. Standards in the IETF are called *Request For Comment* (RFC) and can be written by pretty much anyone who wants to write a standard.

> *To reinforce that we do not vote, we have also adopted the tradition of "humming": When, for example, we have face-to-face meetings and the chair of the working group wants to get a "sense of the room", instead of a show of hands, sometimes the chair will ask for each side to hum on a particular question, either "for" or "against".*
>
> —RFC 7282 ("On Consensus and Humming in the IETF," 2014)

Sometimes, a company publishes a standard directly. For example, RSA Security LLC (funded by the creators of the RSA algorithm) released a series of 15 documents called the *Public Key Cryptography Standards* (PKCS) to legitimize algorithms and techniques the company used at that time. Nowadays, this is pretty rare, and a lot of companies go through the IETF to standardize their protocols or algorithms as an RFC instead of a custom document.

- Your algorithm or protocol gets implemented, go to step 9.
- Nobody cares about your standard, go back to the start.

Step 8: A patent expires. A patent in cryptography usually means that nobody will use the algorithm. Once the patent expires, it is not uncommon to see a renewed interest in the primitive. The most popular example is probably Schnorr signatures, which were the first contender to become the most popular signature scheme until Schnorr himself patented the algorithm in 1989. This led to the NIST standardizing a poorer algorithm called Digital Signature Algorithm (DSA), which became the go-to signature scheme at the time, but doesn't see much use nowadays. The patent over Schnorr signatures expired in 2008, and the algorithm has since started regaining popularity.

- It's been too long, your algorithm will be forever forgotten. Go back to the beginning.
- Your construction inspires many more constructions to get invented on top of it, go to step 5.
- Now people want to use your construction, but not before it's standardized for real. Go to step 7.
- Some developers are implementing your algorithm! Go to step 9.

Step 9: A construction or protocol gets implemented. Implementers have the hard task to not only decipher a paper or a standard (although standards are *supposed* to target implementers), but they also must make their implementations easy and safe to use. This is not always a simple task as many devastating bugs can arise in the way cryptography is used.

- Someone decides it is time for these implementations to be backed by a standard. It's embarrassing without one. Go to step 7.
- Hype is raining on your cryptographic library! Go to step 10.

Step 10: A developer uses a protocol or primitive in an application. A developer has a need, and your cryptographic library seems to solve it—easy peasy!

- The primitive solves the need, but it doesn't have a standard. Not great. Go to step 7.
- I wish this was written in my programming language. Go to step 9.
- I misused the library or the construction is broken. Game over.

You got it! There are many means for a primitive to go real-world. The best way involves many years of analysis, an implementor-friendly standard, and good libraries. A worse way involves a bad algorithm with a poor implementation. In figure 1.16, I illustrate the preferred path.

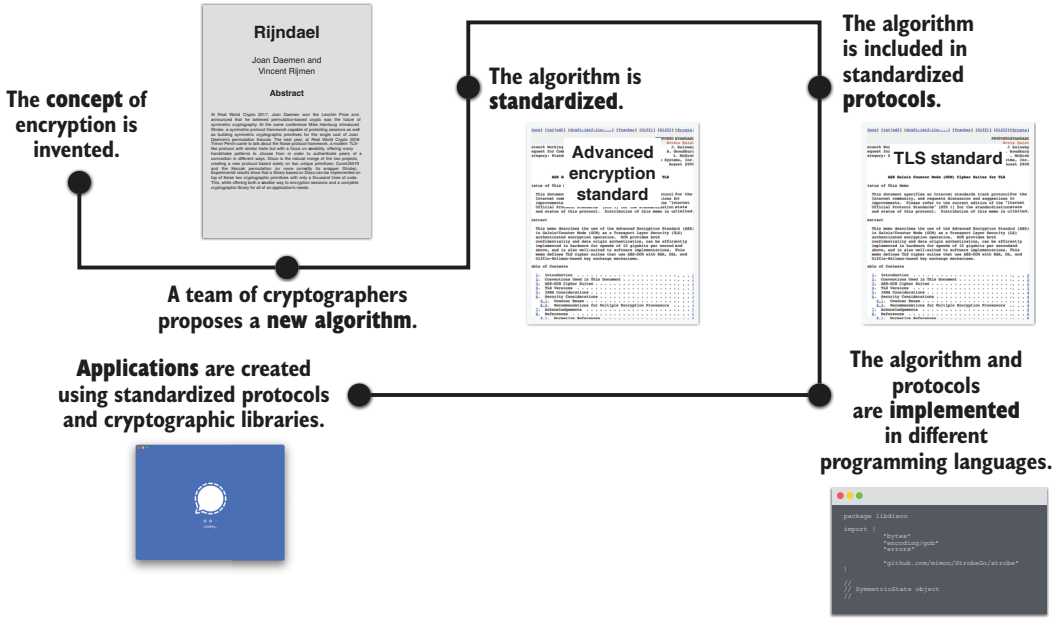


Figure 1.16 The ideal life cycle for a cryptographic algorithm starts when cryptographers instantiate a concept in a white paper. For example, AES is an instantiation of the concept of symmetric encryption (there are many more symmetric encryption algorithms out there). A construction can then be standardized: everybody agrees to implement it a certain way to maximize interoperability. Then support is created by implementing the standard in different languages.

1.8 A word of warning

> *Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break.*
>
> —Bruce Schneier ("Memo to the Amateur Cipher Designer," 1998)

I must warn you, the art of cryptography is difficult to master. It would be unwise to assume that you can build complex cryptographic protocols once you're done with this book. This journey should enlighten you, show you what is possible, and show you how things work, but it will not make you a master of cryptography.

This book is not the holy grail. Indeed, the last pages of this book take you through the most important lesson—do not go alone on a real adventure. Dragons can kill, and you need some support to accompany you in order to defeat them. In other words, cryptography is complicated, and this book alone does not permit you to abuse what you learn. To build complex systems, experts who have studied their trade for years are required. Instead, what you will learn is to recognize when cryptography should be used, or, if something seems fishy, what cryptographic primitives and protocols are available to solve the issues you're facing, and how all these cryptographic algorithms work under the surface. Now that you've been warned, go to the next chapter.

Summary

- A protocol is a step-by-step recipe where multiple participants attempt to achieve something like exchanging confidential messages.
- Cryptography is about augmenting protocols to secure them in adversarial settings. It often requires secrets.
- A cryptographic primitive is a type of cryptographic algorithm. For example, symmetric encryption is a cryptographic primitive, while AES is a specific symmetric encryption algorithm.
- One way to classify the different cryptographic primitives is to split them into two types: symmetric and asymmetric cryptography. Symmetric cryptography uses a single key (as you saw with symmetric encryption), while asymmetric cryptography makes use of different keys (as you saw with key exchanges, asymmetric encryption, and digital signatures).
- Cryptographic properties are hard to classify, but they often aim to provide one of these two properties: authentication or confidentiality. Authentication is about verifying the authenticity of something or someone, while confidentiality is about the privacy of data or identities.
- Real-world cryptography matters because it is ubiquitous in technological applications, while theoretical cryptography is often less useful in practice.
- Most of the cryptographic primitives contained in this book were agreed on after long standardization processes.
- Cryptography is complicated, and there are many dangers in implementing or using cryptographic primitives.